NGINX Conf 2018

The official event for all things NGINX

How to Analyze NGINX Configs

Grant Hulegaard

Software Developer | Technical Track | NGINX

About me

- 5 years – Back, Front and everything in between
- 3 years at NGINX
- NGINX Amplify – Monitoring and Analytics
- NGINX Controller – Amplify + Management

Agenda

1

Why should I analyze my configs?

The problem

- NGINX is a high performance, feature rich…
  - Web server
  - Load balancer
  - Reverse proxy
  - Cache
- NGINX configs are powerful and complex.

$\hookrightarrow$ **Complicated**

The problem

85 modules

214 directives

208 variables

But there's more:

- Conditionals (`if`)
- Go-To's (`rewrite`)
- Case statements (`map` and `geo`)
- [redacted]

The problem – Inheritance

```
server {
    listen 80;
    add_header X-Request-ID $request_id
always;

    location / {
        return 200 “index”;
    }

    location /other/ {
        add_header Cache-Control “no-
cache” always;
        return 200 “other”;
    }
}
```

```
GET / HTTP/1.0

HTTP/1.1 200 ok
Server: nginx/x.xx.x
Date: Mon, 24 Sep 2018, 10:15:22 GMT
Content-Type: application/octet-stream
Content-Length: 5
Connection: close
X-Request-ID: 925d955df378b5e369df9a180669f3ca

index
```

```
GET /other/ HTTP/1.0

HTTP/1.1 200 ok
Server: nginx/x.xx.x
Date: Mon, 24 Sep 2018, 10:15:22 GMT
Content-Type: application/octet-stream
Content-Length: 5
Connection: close
X-Request-ID: 925d955df378b5e369df9a180669f3ca
~~Cache-Control: no-cache~~

other
```

The problem – Best practices

```
server {
    listen 80;

    location / {
        include proxy_headers.conf;
        proxy_pass <some_host>;
    }
}
```

```
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $remote_addr;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Forwarded-Port $server_port;
proxy_set_header X-Forwarded-Protocol $scheme;
```

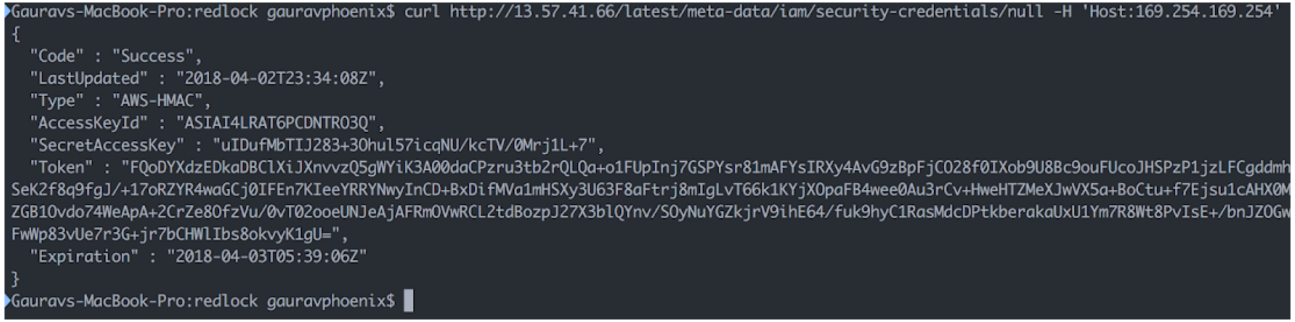
The problem – Bad practices

```
server {
    listen 80;

    location / {
        include
proxy_headers.conf;
        proxy_pass http://
$http_host;
```

$host - In this order of precedence: host name from the request line, host name from the "Host" request header field, or the server name matching a request.

$http_host - host name from the "Host" request header field.

$http_<name> - arbitrary request header field; name is the field name converted to lower case with dashes replaced by underscores.



```
Gauravs-MacBook-Pro:redlock gauravphoenix$ curl http://13.57.41.66/latest/meta-data/iam/security-credentials/null -H 'Host:169.254.169.254'
{
  "Code" : "Success",
  "LastUpdated" : "2018-04-02T23:34:08Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIAI4LRAT6PCDNTRO3Q",
  "SecretAccessKey" : "uIDufMbTIJ283+3Ohul57icqNU/kcTV/0Mrj1L+7",
  "Token" : "FQoDYXdzEDkaDBClXiJXnvvzQ5gWYiK3A00daCPzru3tb2rQLQa+o1FUpInj7GSPYsr81mAFYsIRXy4AvG9zBpFjCO28f0IXob9U8Bc9ouFUcoJHSPzP1jzLFCgddmh
SeK2f8q9fgJ/+17oRZYR4waGCj0IFEn7KIeeYRRYNwyInCD+BxDifMVa1mHSXy3U63F8aFtrj8mIgLvT66k1KYjXOpaFB4wee0Au3rCv+HweHTZMeXJwVX5a+BoCtu+f7Ejsu1cAHX0M
ZGB1Ovdo74WeApA+2CrZe8OfzVu/0vT02ooeUNJeAjAFRmOVwRCL2tdBozpJ27X3blQYnv/SOyNuYGZkjrV9ihE64/fuk9hyC1RasMdcDPtkberakaUxU1Ym7R8Wt8PvIsE+/bnJZOGw
FwWp83vUe7r3G+jr7bCHWlIbs8okvyK1gU=",
  "Expiration" : "2018-04-03T05:39:06Z"
}
Gauravs-MacBook-Pro:redlock gauravphoenix$
```

https://redlock.io/blog/instance-metadata-api-a-modern-day-trojan-hors

We’re not the only ones…

- There are some other open source tools which do *some* analysis of NGINX configs.

2

Parsing NGINX configs

Parsing

- Before we can analyze a config, we need to be able to understand NGINX configs.

    Parse – In linguistics, to divide language into small components that can be analyzed [1].

[1] https://www.webopedia.com/TERM/P/parse.html

A brief history

- NGINX itself does not have a config parser.
- https://github.com/fatiherikli/nginxparser
  - Uses `pyparsing` to parse NGINX configs
  - Slow, susceptible to edge cases.
- Paul McGuire optimized our usage of `pyparsing` for ~**59%** performance improvement.
- Still expensive and we continued to find edge cases two years later. [1]

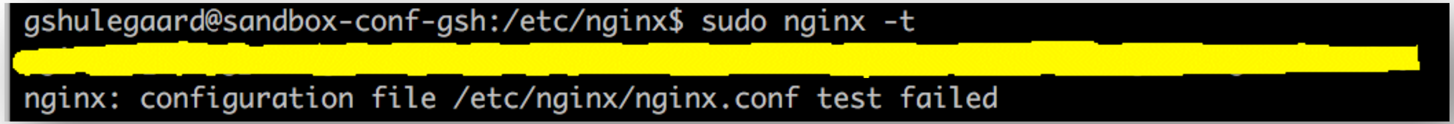
[1] https://github.com/nginxinc/nginx-amplify-agent/blob/484f20a902ed07dc4b50107c0ad6c5d7f14e4681/amplify/agent/objects/nginx/config/parser.py

Crossplane

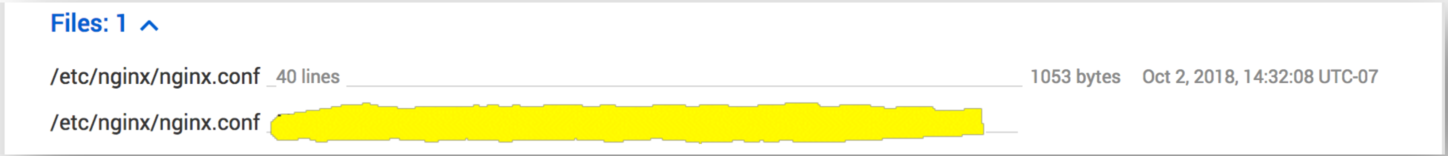
- So we revisit tooling…not much has changed.
- NGINX Config to JSON (and back).
- Faster and much more reliable.
- Even uses NGINX directive definitions for validation. [1]
- https://github.com/nginxinc/crossplane

[1] https://github.com/nginxinc/crossplane/blob/master/crossplane/analyzer.py

NGINX directive definitions



```
gshulegaard@sandbox-conf-gsh:/etc/nginx$ sudo nginx -t
[redacted]
nginx: configuration file /etc/nginx/nginx.conf test failed
```



Files: 1

```
/etc/nginx/nginx.conf  40 lines                    1053 bytes   Oct 2, 2018, 14:32:08 UTC-07
/etc/nginx/nginx.conf  [redacted]
```

simple.conf

```
events {
    worker_connections 1024;
}

http {
    server {
        listen
127.0.0.1:8080;
        server_name
default_server;
        location / {
            return 200 “foo
bar baz”;
        }
    }
}
```

```
{
  "status":"ok",
  "errors":[],
  "config": [
    {
      "file":“simple.conf",
      "status":"ok",
      "errors":[],
      "parsed":[
        {
          "directive":"events",
          "args":[],
          "line":1,
          "block":[
            {
"directive":"worker_connections“
              "args":["1024"],
              "line":2,
            }
          ]
        },
        …
]}]}]}
```

```
        {
          "directive":"http",
          "args":[],
          "line":5,
          "block":[
            {
              "directive":"server",
              "args":[],
              "line":6,
              "block":[
                {
                  "directive":"listen",
                  "args":
["127.0.0.1:8080"],
                  "line":7
                },
                {
"directive":"server_name",
                  "args":
["default_server"],
                  "line":8,
                },
                {
                  "directive":"location",
                  "args":["/"],
                  "line":9,
                  "block":[
                    {
"directive":"return",
                      "args":["200","foo
bar baz"],
                      "line":10
                    }
                  ]
                }
              ]
            }
        }
```

3

A few examples

Inheritance

```
server {
    listen 80;
    add_header X-Request-ID $request_id
always;

    location / {
        return 200 “index”;
    }

    location /other/ {
        add_header Cache-Control “no-cache”
always;
        return 200 “other”;
    }
}
```

```
…
block: [
  {“directive”: ”add_header”, …},
  …
  {
    “directive”: “location”,
    “args”: [“/other/“],
    …
    “block”: [
      {“directive”: “add_header”, …},
      …
    ]
  },
  …
]
```

Inheritance

- Initialize a flag (`False`).
- Whenever you find an `add_header` directive consult the flag.
  - If the flag is `False`, flip it and continue.
  - If the flag is already `True`, you've found an inheritance rewrite.

```
…                                                 False
block: [
  …
  {“directive”: ”add_header”, …},                 True
  …
  {
    “directive”: “location”,
    “args”: [“/other/“],
    …
    “block”: [
      {“directive”: “add_header”, …},             Warn
      …
    ]
  },
  …
]
```

Best practices

```
server {
    listen 80;

    location / {
        include proxy_headers.conf;
        proxy_pass <some_host>;
    }
}
```

- Initialize a “context”.
- When you find `proxy_set_header`, overwrite the context.
- When you find `proxy_pass`, consult the context.

Bad practices

```
server {
    listen 80;

    location / {
        include proxy_headers.conf;
        proxy_pass http://$http_host;
    }
}
```

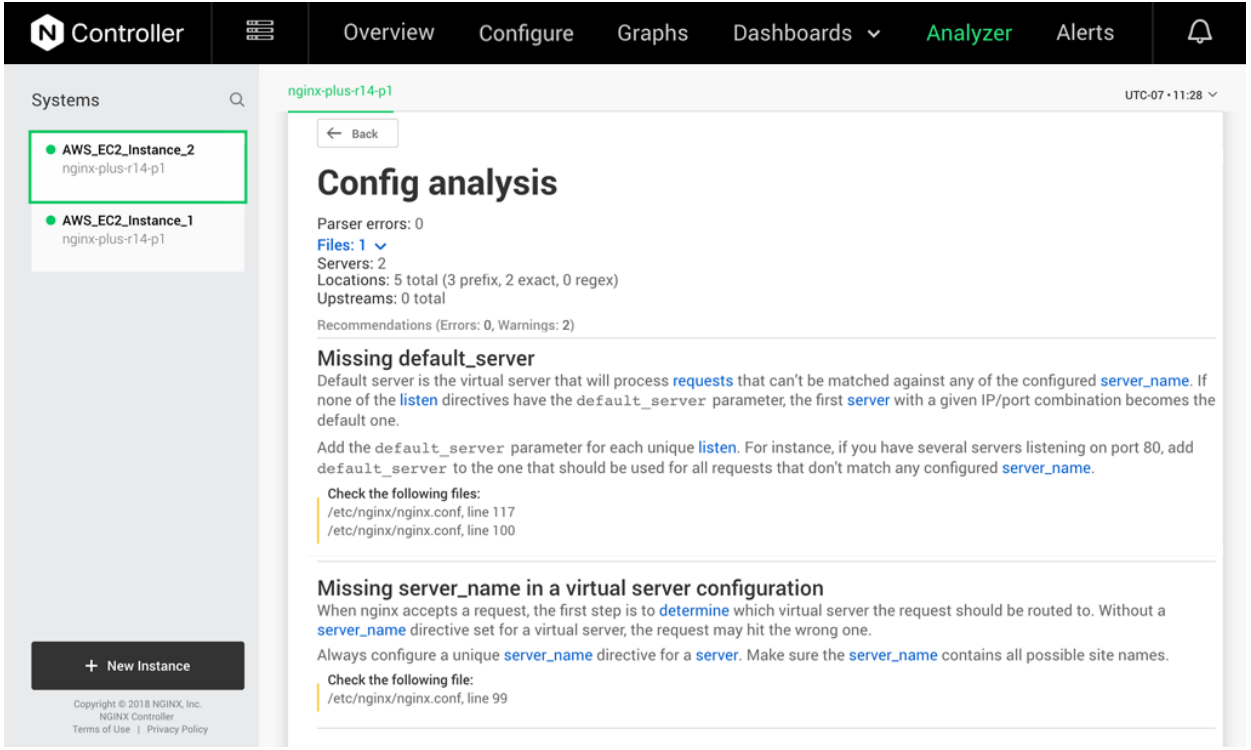
```
...
{
  “directive”: “proxy_pass”,
  “args”: [“http://$http_host”],
  …
}
```

Bad practices

- Look for `proxy_pass` directives.
- Check the first arg...
- Split by "/" and check if the first variable or the first variable after scheme is vulnerable.
  - http://$http_host/ ✗
  - $host/ ✗
  - http://10.10.10.54/?arg=$http_host ✓

```
...
{
  "directive": "proxy_pass",
  "args": ["http://$http_host"],
  …
}
```

NGINX Controller



Copyright © 2018 NGINX, Inc.
NGINX Controller
Terms of Use | Privacy Policy

Config analysis

Parser errors: 0
Files: 1
Servers: 2
Locations: 5 total (3 prefix, 2 exact, 0 regex)
Upstreams: 0 total

Recommendations (Errors: 0, Warnings: 2)

Missing default_server

Default server is the virtual server that will process requests that can't be matched against any of the configured server_name. If none of the listen directives have the `default_server` parameter, the first server with a given IP/port combination becomes the default one.

Add the `default_server` parameter for each unique listen. For instance, if you have several servers listening on port 80, add `default_server` to the one that should be used for all requests that don't match any configured server_name.

Check the following files:
/etc/nginx/nginx.conf, line 117
/etc/nginx/nginx.conf, line 100

Missing server_name in a virtual server configuration

When nginx accepts a request, the first step is to determine which virtual server the request should be routed to. Without a server_name directive set for a virtual server, the request may hit the wrong one.

Always configure a unique server_name directive for a server. Make sure the server_name contains all possible site names.

Check the following file:
/etc/nginx/nginx.conf, line 99

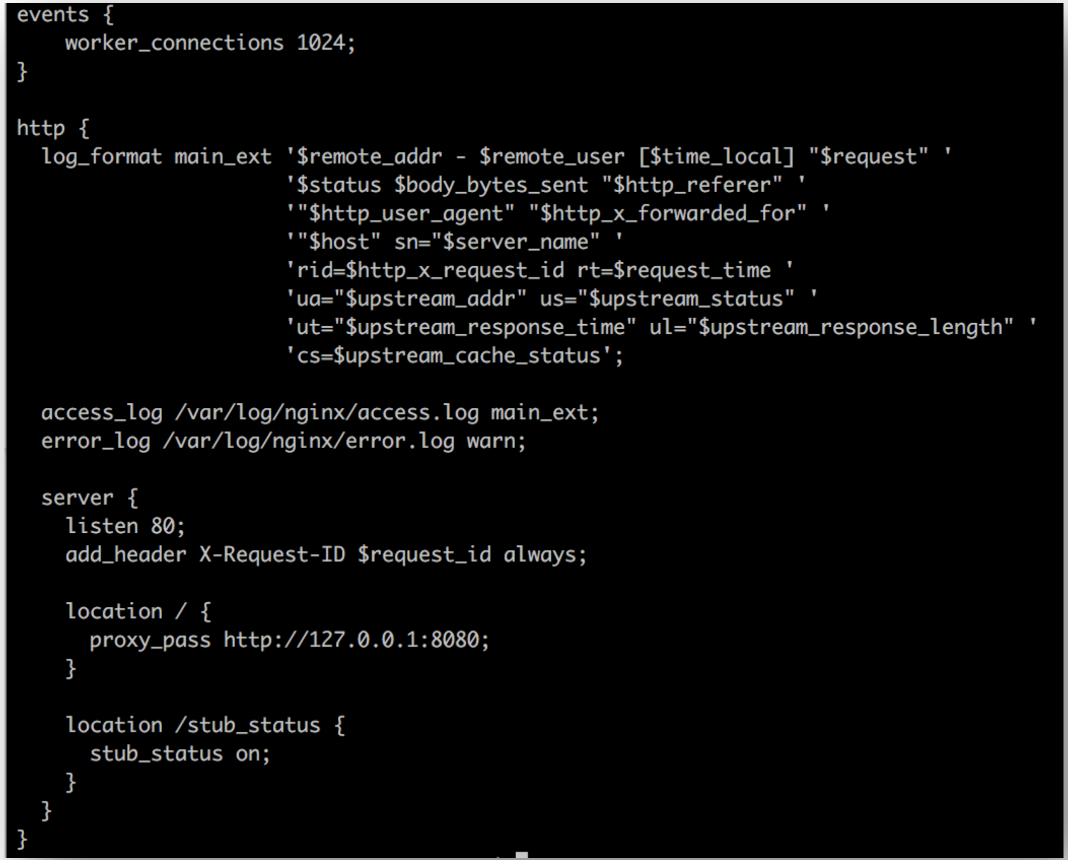
Currently available reports

- **Proxy pass headers**
- Missing server names
- Worker processes
- Missing default conf
- Proxy buffering
- Proxy request buffering
- Missing default server
- Regex checks
- Stub status ACL
- Plus status ACL
- Plus API ACL
- Dangerous rewrite checks
- Wildcard TCP socket overlaps

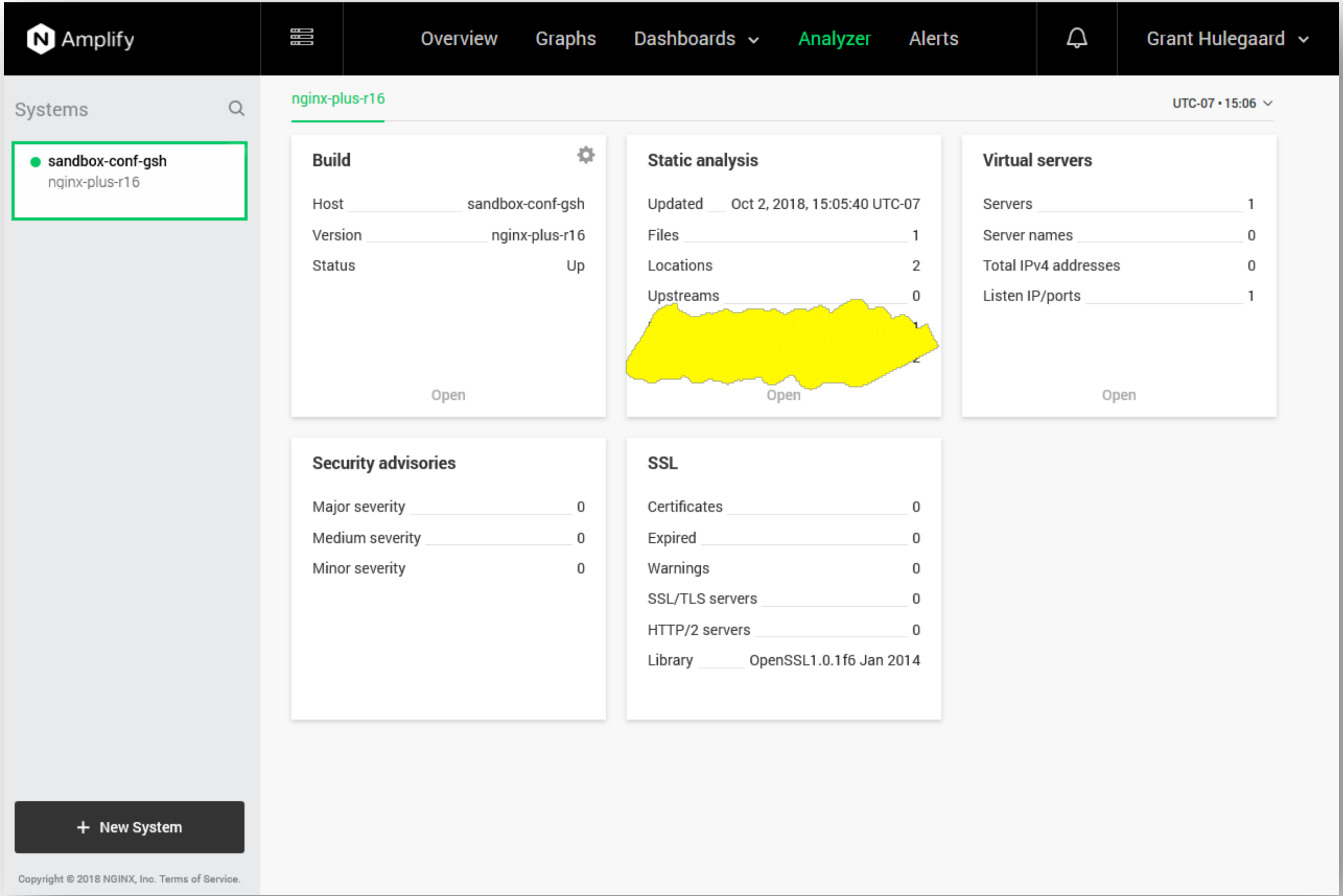
- FastCGI params
- Listen overlaps
- Obsolete SSL configuration
- Missing listens
- Missing error log
- Multiple stub status
- SSL protocol checks
- Missing SSL protocol
- SSL cipher checks
- **Insecure proxy pass**
- **Header inheritance overlap**
- Debug connection without –with-debug

- Configuration overview
- SSL certificate expiration
- SSL certificate info checks
- NGINX security advisories

A live example



```
events {
    worker_connections 1024;
}

http {
  log_format main_ext '$remote_addr - $remote_user [$time_local] "$request" '
                      '$status $body_bytes_sent "$http_referer" '
                      '"$http_user_agent" "$http_x_forwarded_for" '
                      '"$host" sn="$server_name" '
                      'rid=$http_x_request_id rt=$request_time '
                      'ua="$upstream_addr" us="$upstream_status" '
                      'ut="$upstream_response_time" ul="$upstream_response_length" '
                      'cs=$upstream_cache_status';

  access_log /var/log/nginx/access.log main_ext;
  error_log /var/log/nginx/error.log warn;

  server {
    listen 80;
    add_header X-Request-ID $request_id always;

    location / {
      proxy_pass http://127.0.0.1:8080;
    }

    location /stub_status {
      stub_status on;
    }
  }
}
```



sandbox-conf-gsh
nginx-plus-r16

nginx-plus-r16 UTC-07 • 15:06

Build

Host	sandbox-conf-gsh
Version	nginx-plus-r16
Status	Up

Open

Static analysis

Updated	Oct 2, 2018, 15:05:40 UTC-07
Files	1
Locations	2
Upstreams	0

[redacted]

Open

Virtual servers

Servers	1
Server names	0
Total IPv4 addresses	0
Listen IP/ports	1

Open

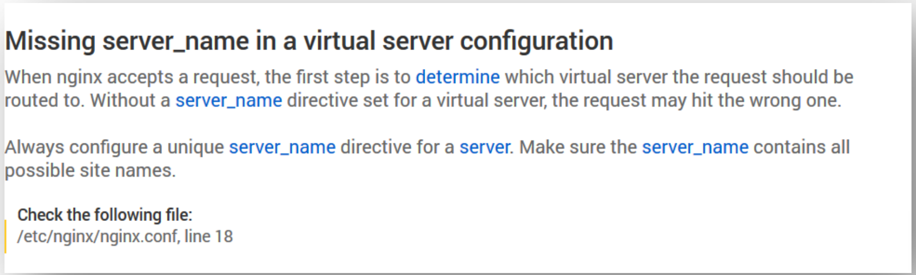
Security advisories

Major severity	0
Medium severity	0
Minor severity	0

SSL

Certificates	0
Expired	0
Warnings	0
SSL/TLS servers	0
HTTP/2 servers	0
Library	OpenSSL1.0.1f6 Jan 2014

Copyright © 2018 NGINX, Inc. Terms of Service.

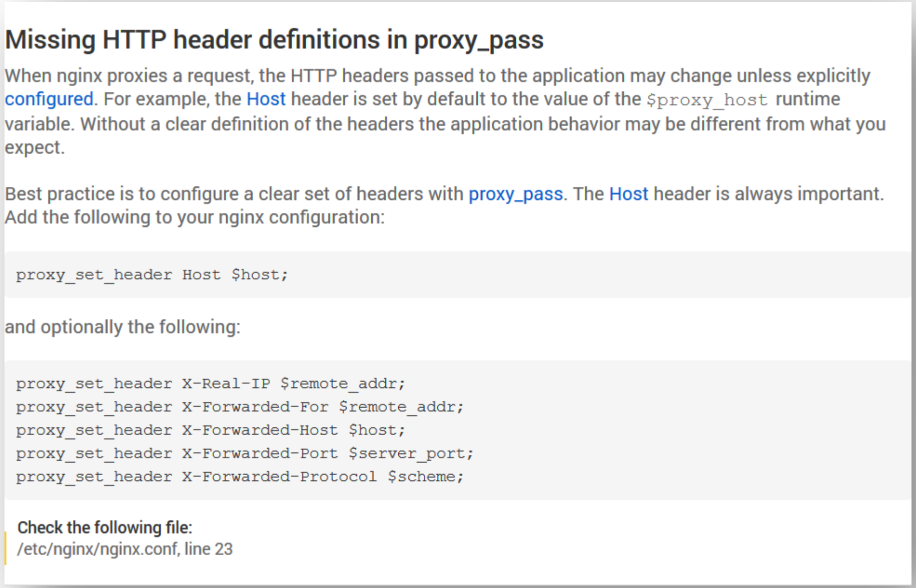


Missing server_name in a virtual server configuration

When nginx accepts a request, the first step is to determine which virtual server the request should be routed to. Without a server_name directive set for a virtual server, the request may hit the wrong one.

Always configure a unique server_name directive for a server. Make sure the server_name contains all possible site names.

Check the following file:
/etc/nginx/nginx.conf, line 18



Missing HTTP header definitions in proxy_pass

When nginx proxies a request, the HTTP headers passed to the application may change unless explicitly configured. For example, the Host header is set by default to the value of the `$proxy_host` runtime variable. Without a clear definition of the headers the application behavior may be different from what you expect.

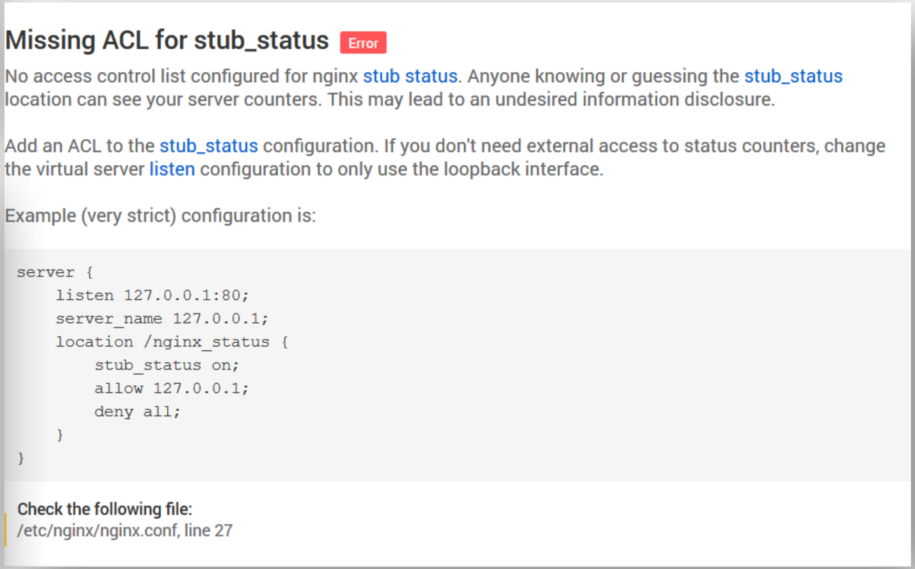
Best practice is to configure a clear set of headers with proxy_pass. The Host header is always important. Add the following to your nginx configuration:

```
proxy_set_header Host $host;
```

and optionally the following:

```
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $remote_addr;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Forwarded-Port $server_port;
proxy_set_header X-Forwarded-Protocol $scheme;
```

Check the following file:
/etc/nginx/nginx.conf, line 23



Missing ACL for stub_status Error

No access control list configured for nginx stub status. Anyone knowing or guessing the stub_status location can see your server counters. This may lead to an undesired information disclosure.

Add an ACL to the stub_status configuration. If you don't need external access to status counters, change the virtual server listen configuration to only use the loopback interface.

Example (very strict) configuration is:

```
server {
    listen 127.0.0.1:80;
    server_name 127.0.0.1;
    location /nginx_status {
        stub_status on;
        allow 127.0.0.1;
        deny all;
    }
}
```

Check the following file:
/etc/nginx/nginx.conf, line 27

4

Wrap-up

Wrap-up

- Talked about why you would want to analyze NGINX configs.
- Parsed configs with `crossplane`.
- Walked through some example analyses.
- Talked about analysis in NGINX Controller today.
- Used the analyzer with a live example.

NGINX

Thank you

grant.hulegaard@nginx.com

@gshulegaard